1. Метод розв'язання

Для розв'язання СЛАУ використаємо ітераційний метод простої ітерації.

Метод простої ітерації являє собою модифікацію метода послідовних наближень, при чому у методі простої ітерації при обчисленні і-ої координати вектора розв'язку $(k+1)$ - го наближення використовуються значення всіх (і-1) координат вектора $(k+1)$ - е наближення обчислені раніше. Розглянемо метод більш детально.

Дана система линенйных алгебраических уравнений вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \quad \ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

$$Ax = b$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1(n-1)} & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2(n-1)} & a_{2n} \\ . & . & \ldots & . & . \\ a_{(n-1)1} & a_{(n-1)2} & \ldots & a_{(n-1)(n-1)} & a_{(n-1)n} \\ a_{n1} & a_{n2} & \ldots & a_{n(n-1)} & a_{nn} \end{pmatrix} b = \begin{pmatrix} b_1 \\ b_2 \\ . \\ b_{n-1} \\ b_n \end{pmatrix} x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_{n-1} \\ x_n \end{pmatrix}$$

Предполагается, что диагональные коэффициенты ненулевые.

$$a_{ii} \neq 0 \qquad i = (1,2,..,n)$$

Решив 1-ое уравнение системы относительно x_1 получим:

$$x_1 = \frac{b_1 - (a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n)}{a_{11}}$$

2-ое относительно x_2 , n-ое относительно x_n

В итоге эквивалентная система, в которой диагональные элементы строки выражены через оставщиеся.

$$\begin{cases} x_1 = \dfrac{b_1 - (a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n)}{a_{11}} \\ x_2 = \dfrac{b_2 - (a_{21}x_1 + a_{23}x_3 + \ldots + a_{2n}x_n)}{a_{22}} \\ \quad \ldots \\ x_n = \dfrac{b_n - \left(a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{n(n-1)}x_{n-1}\right)}{a_{nn}} \end{cases}$$

Из системы выделим матрицы.

$$\alpha = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & \ldots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \ldots & -\frac{a_{2n}}{a_{22}} \\ & & \ldots & & \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & -\frac{a_{n3}}{a_{nn}} & \ldots & 0 \end{pmatrix} \qquad \beta = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ . \\ \frac{b_n}{a_{nn}} \end{pmatrix}$$

Вводится некоторое начальное приближение - вектор x(0)(значения задаются произвольно либо дается вектор свободных членов или нулевые значения). Подставив x(0) в полученную систему находится первое приближение x(1), затем используях(1) находится x(2).

Условием окончания является достижение заданной точности(система сходится и есть решение) или прерывание процесса. Процесс прерывается когда число итераций превышает заданное допустимое количество(защита от "взрыва"), при этом система не сходится либо заданное количество итераций не хватило для достижения требуемой точности. Итерационный процесс. Верхний индекс в скобках - номер итерации. Учитываются уже вычисленные ранее приближения.

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1 - \left(a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \ldots + a_{1n}x_n^{(k)}\right)}{a_{11}} \\ x_2^{(k+1)} = \dfrac{b_2 - \left(a_{21}x_1^{(k+1)} + a_{23}x_3^{(k)} + \ldots + a_{2n}x_n^{(k)}\right)}{a_{22}} \\ x_3^{(k+1)} = \dfrac{b_3 - \left(a_{31}x_1^{(k+1)} + a_{32}x_3^{(k+1)} + \ldots + a_{2n}x_n^{(k)}\right)}{a_{22}} \\ \ldots \\ x_n^{(k+1)} = \dfrac{b_n - \left(a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + \ldots + a_{n(n-1)}x_{n-1}^{(k+1)}\right)}{a_{nn}} \end{cases}$$

В итоге получается последовательность приближений $\left(x^{(0)}, x^{(1)}, \ldots, x^{(k+1)}, \ldots\right)$, и если эта последовательность имеет предел

$$x = \lim_{k \to \infty} x^{(k)}$$

то этот предел является решением.

$k = 1,2,3,\ldots N\text{-}1.$, N-1 - заданное количество итераций

Достаточный признак сходимости метода Якоби выполняется и для метода Зейделя:
Если в системе выполняется диагональное преобладание, то метод сходится.
То есть в каждой строке диагональный элемент по модулю больше суммы модулей остальных элементов. Если данное условие не выполняется, необходимо соответствующим образом преобразовать СЛАУ, привести к удобному для итерации виду, выполнив эквивалентные преобразования (перестановка строк, линейная комбинация строк).

$$|a_{ii}| > \sum_{j=1,(j \neq i)}^{n} |a_{ij}| \qquad i = 1,2,..,n$$

2. Лістинг

```
#include <iostream>
#include <fstream>
#include <cmath>

#define n 5
#define eps 0.00001

using namespace std;


double Residual(double** A, double* x, int k){
	double *r=new double[k], tmp, norm;
	for(int i=0; i<k; i++){
		tmp=0;
		for(int j=0; j<k; j++)
			tmp+=A[i][j]*x[j];
		r[i]=tmp-A[i][k];
```

```
        }
        cout << "Vector of residual: "<< endl;
        for (int i=0; i<k; i++){
                cout.width(15);
                cout << r[i];
        }

        norm = 0;
        for(int i = 0; i < k; i++) norm = norm + r[i]*r[i];
        norm = sqrt(norm);
        cout<<"\nNorma: "<<norm<<"\n\n";

        return norm;
}

int simple_iteration(double **mat, double *res, int m){

    int i, j;

    //check for diag. ..
    double temp_sum_of_elem;
    for(i = 0; i < m; i++){
        temp_sum_of_elem=0;
                for(j = 0; j < m; j++)    temp_sum_of_elem+=mat[i][j];
                temp_sum_of_elem-=mat[i][i];
                if( fabs(mat[i][i]) <= fabs(temp_sum_of_elem) ) return 1;
        }

    //check for zeros on the diagonal
        for(i = 0; i < m; i++) {
            if(mat[i][i] == 0) return 1;
    }

        //install values for free values (the last one column in our matrix)
        double *free_vec = new double[m];
        for(i = 0; i < m; i++)
            free_vec[i] = mat[i][m]/mat[i][i];

        //directly our cycle
        double *temp = new double[m];
        double sum;
        double norm;
        int count = 0;

        do {
```

```
            //resend values of results to temporary array to use 'em
later/again&again
        for(i = 0; i < m; i++)
            temp[i] = res[i];

        //calculation of the values of x^i
            for(i = 0; i < m; i++){
        sum = 0;
                for(j = 0; j < m; j++) if(i != j) sum -= res[j]*mat[i][j]/mat[i][i];
                res[i] = sum + free_vec[i];
    }

        //restriction on the number of iterations
        count++;
        if(count > 10000) return 0;

            //calculation of the norm of residual
            norm = 0;
            for(i = 0; i < m; i++){
    norm = norm + (res[i] - temp[i])*(res[i] - temp[i]);
            }
            norm = sqrt(norm);

            cout<<"Iteration "<<count<<".\n";
            cout<<"Roots: { ";
            for (i=0;i<m-1;i++) cout<<res[i]<<", ";
            cout<<res[m-1]<<" }\n";
            norm=Residual(mat, res, m);

    } while(norm > eps);

    return 0;
}

int main() {
    int i, j, k;
    //results initialisation
    double *result = new double[n];

    //matrix initialization
    double **A = new double*[n];
    for(i = 0; i < n; i++)
        A[i] = new double[n + 1];

    //read matrix from "matrix.dat"
    ifstream input_file("matrix.dat");
```

```
        char temp[10];
        for(i = 0; i < n; i++)
                for(j = 0; j < n + 1; j++){
                                input_file>>A[i][j];
                                if (input_file.fail()) return 0;
                }
        input_file.close();

        //matrix output
        cout << "Matrix A|b: \n";
        for(i = 0; i < n; i++){
            for(j = 0; j < n + 1; j++)
                cout << A[i][j] << " ";
            cout << "\n";
    }
    cout <<"\n";

        //initial values of the results
        for(i = 0; i < n; i++)
            result[i] = A[i][n]/A[i][i];

        //execute iteration method
        if(simple_iteration(A, result, n) != 0) cout << "\nError!";
        else {
                    cout << "\nRoots: ";
                    for(i = 0; i < n; i++)
                   cout << result[i] << " ";
    }
        return 0;
}
```

3. Результати роботи

```
Matrix A|b:
6.71 1.16 0.91 1.18 -0.36 2.1
1.04 3.72 1.3 -1.63 0.12 0.48
1.03 -2.46 5.88 2.1 0.583 1.29
0.84 -0.78 -0.317 3 1 -0.48
1.3 0.16 2.1 5.66 -6 6.44

Iteration 1.
Roots: { 0.231457, -0.0478279, 0.322397, 0.154601, -0.76578 }
Vector of residual:
     0.149091      -0.341981       0.839966       0.307553   8.88178e-016
Norma: 0.96918

Iteration 2.
Roots: { 0.209238, 0.0503143, 0.224498, 0.073477, -0.878769 }
Vector of residual:
    -0.0302941    -0.00859601      -0.236233      -0.112989   8.88178e-016
Norma: 0.26375
```

Iteration 3.
Roots: { 0.213753, 0.0513629, 0.264321, 0.114356, -0.825262 }
Vector of residual:
0.0664309 -0.0084421 0.117042 0.0535074 0
Norma: 0.145073

Iteration 4.
Roots: { 0.203853, 0.0564001, 0.248258, 0.0989051, -0.847471 }
Vector of residual:
-0.0190121 0.0016383 -0.0453957 -0.0222088 8.88178e-016
Norma: 0.0540198

Iteration 5.
Roots: { 0.206686, 0.0551675, 0.254966, 0.105903, -0.83794 }
Vector of residual:
0.00950154 -0.00154215 0.0202519 0.00953038 8.88178e-016
Norma: 0.0243645

Iteration 6.
Roots: { 0.20527, 0.055978, 0.252109, 0.103032, -0.841934 }
Vector of residual:
-0.0036104 0.000487026 -0.00835861 -0.00399395 8.88178e-016
Norma: 0.00995441

Iteration 7.
Roots: { 0.205808, 0.0556966, 0.253318, 0.104267, -0.840236 }
Vector of residual:
0.00162084 -0.0002374 0.00358396 0.00169775 8.88178e-016
Norma: 0.00429076

Iteration 8.
Roots: { 0.205567, 0.055828, 0.252806, 0.103749, -0.840953 }
Vector of residual:
-0.000667205 9.27909e-005 -0.00150636 -0.000717013 8.88178e-016
Norma: 0.00179917

Iteration 9.
Roots: { 0.205666, 0.0557752, 0.253023, 0.103969, -0.84065 }
Vector of residual:
0.000286642 -4.09952e-005 0.000639871 0.000303844 8.88178e-016
Norma: 0.000765245

Iteration 10.
Roots: { 0.205623, 0.0557982, 0.252931, 0.103876, -0.840778 }
Vector of residual:
-0.000120362 1.69861e-005 -0.000270316 -0.00012852 8.88178e-016
Norma: 0.000323054

Iteration 11.
Roots: { 0.205641, 0.0557886, 0.25297, 0.103915, -0.840724 }
Vector of residual:
5.11507e-005 -7.27062e-006 0.000114522 5.44127e-005 8.88178e-016
Norma: 0.000136914

Iteration 12.

```
Roots: { 0.205634, 0.0557927, 0.252954, 0.103899, -0.840747 }
Vector of residual:
  -2.16034e-005   3.05912e-006  -4.84462e-005   -2.3026e-005  -8.88178e-016
Norma: 5.79077e-005

Iteration 13.
Roots: { 0.205637, 0.055791, 0.252961, 0.103906, -0.840737 }
Vector of residual:
   9.15372e-006  -1.29873e-006   2.05101e-005   9.74651e-006   8.88178e-016
Norma: 2.4518e-005

Iteration 14.
Roots: { 0.205636, 0.0557917, 0.252958, 0.103903, -0.840741 }
Vector of residual:
  -3.87202e-006   5.48805e-007  -8.67958e-006  -4.12497e-006              0
Norma: 1.03752e-005

Iteration 15.
Roots: { 0.205636, 0.0557914, 0.252959, 0.103904, -0.840739 }
Vector of residual:
   1.63931e-006  -2.32473e-007   3.67386e-006   1.74592e-006  -8.88178e-016
Norma: 4.39168e-006


Roots: 0.205636 0.0557914 0.252959 0.103904 -0.840739
--------------------------------
Process exited after 0.1269 seconds with return value 0
Press any key to continue . . .
```